UNIVERSITATEA “STEFAN CEL MARE”, SUCEAVA

FACULTATEA DE INGINERIE ELECTRICA SI STIINTA CALCULATOARELOR

SPECIALIZAREA CALCULATOARE

PROIECT
DISCIPLINA POO

Tema: Parseline – Interfață CLI cu parsing de comenzi și REPL

Autor: Mihail Gosman

Tema Proiect

TEMA SI MOTIVATIA ALEGERII

Acest proiect constă în realizarea unui interpretor de linie de comandă (CLI) denumit „Parseline”, care acceptă comenzi text, le parsează folosind expresii regulate, le execută și păstrează istoricul comenzilor introduse. Aplicația permite adăugarea ușoară de noi comenzi, oferă sugestii, istoric și ajutor.

Motivația a fost dorința de a crea o interfață de comandă extensibilă, utilă în aplicații ce necesită interactivitate și procesare flexibilă a inputului textual.

CUPRINS

TEMA ȘI MOTIVATIA ALEGERII

CAPITOLUL I – ELEMENTE TEORETICE

1.1. DESCRIEREA PROBLEMEI

Parseline este o bibliotecă C++ destinată construirii rapide și simple a aplicațiilor de tip linie de comandă (CLI). Aceasta permite definirea de comenzi personalizate cu ajutorul expresiilor regulate, parsarea argumentelor și opțiunilor, memorarea istoricului comenzilor, oferirea de sugestii pentru comenzi necunoscute și afișarea mesajelor de ajutor.

1.2. ABORDAREA TEORETICA A PROBLEMEI

Pentru a construi un astfel de sistem, problema trebuie împărțită în mai multe subprobleme fundamentale:

Parsarea liniei de comandă

- **Input:** O linie de text introdusă de utilizator.
- **Output:** O structură organizată ce conține:
    - Comanda principală (de ex. sum)
    - Argumentele comenzii (de ex. 5, 10)
    - Opțiuni (de ex. --verbose, -h)

Parsarea presupune identificarea corectă a componentelor comenzii, respectând convențiile uzuale pentru argumente și opțiuni.

Recunoașterea și validarea comenzilor

- Sistemul trebuie să recunoască dacă comanda introdusă este validă și există un handler asociat.
- Validarea poate include verificarea sintaxei (ex: număr corect de argumente), dar și potrivirea pattern-ului comenzii.

- Pattern-urile pot fi exprimate cu expresii regulate, pentru a permite flexibilitate și combinarea de condiții.

Executarea comenzilor

- Pentru fiecare comandă recunoscută, trebuie definit un handler (funcție) care primește argumentele parsate și execută logica aferentă.

- Handlerul trebuie să aibă acces la argumente și opțiuni în mod ușor.

Gestionarea istoricului comenzilor

- Sistemul trebuie să păstreze o listă a comenzilor introduse anterior pentru:

    - Vizualizare ulterioară (comanda history)

    - Eventuală reutilizare sau editare

Suport pentru ajutor și sugestii

- Ajutor contextual: afișarea unor mesaje explicative pentru comenzi, astfel încât utilizatorul să știe cum să folosească aplicația.

- Sugestii: în cazul unei comenzi necunoscute, sistemul poate sugera comenzi similare (bazat pe prefix, similaritate, etc.)

1.3. ELEMENTE SPECIFICE POO

În proiect vor fi aplicate elemente fundamentale ale programării orientate pe obiect, cum ar fi:

- **Încapsularea:** Atributele claselor vor fi protejate prin metode de acces (getteri/setteri) și funcții dedicate.
- **Moștenirea:** Posibilă extindere a claselor pentru a adăuga funcționalități specifice (de exemplu, clase derivate pentru categorii particulare de laptopuri).

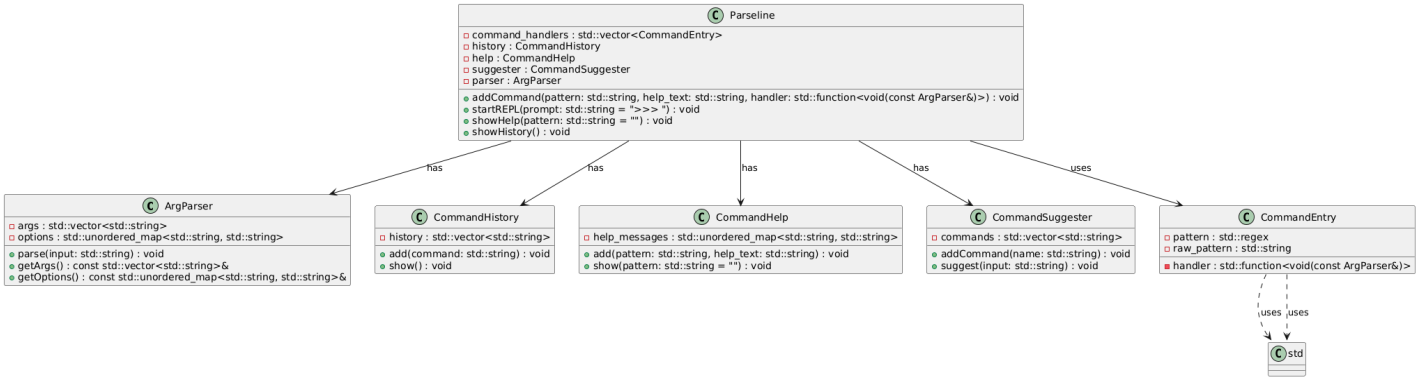
CAPITOLUL II – IMPLEMENTARE

2.1. TEHNOLOGII FOLOSITE

- **Limbajul de programare:** C++
- **Mediu de dezvoltare:** Code::Blocks
- **Biblioteci standard:** iostream, functional, vector, string, regex

2.2. DIAGRAMA DE CLASE

Diagrama UML de mai jos prezintă clasele principale și relațiile dintre ele:



2.3. IMPLEMENTAREA FUNCȚIONALITĂȚILOR SPECIFICE

Funcționalități propuse:

- **Comenzi cu argumente complexe și tipuri multiple**
- **Pipelining**
- **Configurare și personalizare**
- **Extinderea sistemului de ajutor**

CAPITOLUL III – ANALIZA SOLUȚIEI IMPLEMENTATE

3.1. FORMATUL DATELOR DE I/O

Parsarea liniei de comandă (ArgParser)

- **Fișiere implicate:** include/argparser.hpp, src/argparser.cpp
- **Descriere:**

  Clasa ArgParser este responsabilă pentru parsarea input-ului de la utilizator în componentele sale: argumente și opțiuni.

- **Implementare:**
  - Metoda parse(const std::string& input) primește o linie de text.
  - Textul este împărțit în token-uri pe spații folosind std::istringstream.
  - Fiecare token este analizat pentru a identifica:
    - Opțiuni lungi (încep cu --), eventual cu valoare atașată prin semnul egal, de ex. --file=output.txt.
    - Opțiuni scurte (încep cu -), de ex. -v.
    - Argumente normale (fără prefix).
  - Argumentele și opțiunile sunt stocate în vectorul args și respectiv în options (hash map).
- **Avantaje:**
  - Simplitate și eficiență.
  - Ușor extensibil pentru alte formate de opțiuni.

Gestionarea istoricului comenzilor (CommandHistory)

- **Fișiere implicate:** include/command_history.hpp, src/command_history.cpp
- **Descriere:**

  Clasa CommandHistory păstrează un istoric al comenzilor introduse în sesiunea curentă.

- **Implementare:**
  - Comenzile sunt adăugate în vectorul history prin metoda add().
  - Metoda show() afișează lista comenzilor salvate, una pe linie.
  - Istoricul este utilizat în interfața REPL pentru afișarea comenzilor anterioare.

Afișarea mesajelor de ajutor (CommandHelp)

- **Fișiere implicate:** include/command_help.hpp, src/command_help.cpp
- **Descriere:**

  Clasa CommandHelp gestionează mesajele de ajutor asociate comenzilor.

- **Implementare:**
  - Mesajele de ajutor sunt stocate într-un unordered_map cu cheia fiind pattern-ul comenzii.
  - add(pattern, help_text) înregistrează un mesaj de ajutor.
  - show(pattern) afișează fie toate mesajele (dacă pattern-ul este gol), fie doar mesajul corespunzător unei comenzi specifice.

- În cazul în care nu există ajutor pentru o comandă, se afișează un mesaj corespunzător.

Sugestii pentru comenzi (CommandSuggester)

- **Fișiere implicate:** include/command_suggester.hpp, src/command_suggester.cpp
- **Descriere:**

  Sugerează comenzi similare când utilizatorul introduce o comandă necunoscută.

- **Implementare:**
  - Comenzile sunt adăugate într-un vector commands.
  - Metoda suggest(input) verifică comenzile care încep cu textul introdus.
  - Pentru fiecare potrivire, se afișează o sugestie sub forma: "Did you mean: <comanda>?"

Definirea și executarea comenzilor (Parseline)

- **Fișiere implicate:** include/parseline.hpp, src/parseline.cpp
- **Descriere:**

  Clasa Parseline reprezintă nucleul aplicației. Ea gestionează înregistrarea comenzilor, procesarea input-ului, rularea REPL-ului și orchestrarea celorlalte componente.

- **Implementare:**
  - Comenzile sunt adăugate prin addCommand(pattern, help_text, handler), unde:

        - pattern este o expresie regulată pentru recunoașterea comenzii.
        - help_text este mesajul de ajutor.
        - handler este o funcție care primește un obiect ArgParser pentru a accesa argumentele.
    - În metoda startREPL():
        - Se citește linie cu linie inputul utilizatorului.
        - Se stochează în istoric.
        - Se permite ieșirea cu comanda exit.
        - Linia poate conține mai multe comenzi separate prin pipe (|). Acestea sunt separate cu splitPipe().
        - Pentru fiecare comandă, se parsează argumentele și opțiunile.
        - Se caută o comandă care să corespundă expresiei regulate.
        - Dacă se găsește, se apelează handlerul asociat.
        - Dacă nu, se afișează mesajul de eroare și se oferă sugestii.
    - Metode suplimentare:
        - showHelp() afișează mesajele de ajutor.
        - showHistory() afișează istoricul comenzilor.
        - splitPipe() împarte linia pe simbolul pipe, permițând execuția mai multor comenzi într-un singur input.
        - trim() elimină spațiile inutile din jurul textului.

3.2. STUDII DE CAZ

Cazuri reprezentative ce vor fi prezentate:

Utilizarea comenzii simple fără argumente

- **Comanda:** hello
- **Descriere:** Apelarea comenzii „hello” fără argumente va afișa un mesaj de salut implicit.
- **Scop:** Demonstrează funcționarea de bază a interpretatorului de comenzi.

Utilizarea comenzii cu argumente

- **Comanda:** hello <nume>
- **Descriere:** Comanda salută utilizatorul pe baza numelui furnizat ca argument.
- **Scop:** Ilustrează parsing-ul argumentelor și folosirea lor în handler.

Executarea unei comenzi cu mai multe argumente și validare

- **Comanda:** sum <a> <b>
- **Descriere:** Adună două numere întregi transmise ca argumente. Dacă argumentele lipsesc sau nu sunt valide, afișează un mesaj de utilizare.
- **Scop:** Prezintă manipularea argumentelor și mesajele de eroare.

Vizualizarea istoricului comenzilor

- **Comanda:** history
- **Descriere:** Afișează lista comenzilor introduse anterior în sesiunea curentă.
- **Scop:** Demonstrează stocarea și afișarea istoricului comenzilor.

Accesarea ajutorului pentru comenzi

- **Comanda:** help sau help <comandă>
- **Descriere:** Afișează lista comenzilor disponibile sau ajutor specific pentru o comandă anume.
- **Scop:** Ilustrează sistemul de documentare integrat.

Gestionarea comenzilor necunoscute și sugestii

- **Comanda:** comenzi inexistente, ex: helo
- **Descriere:** În cazul unei comenzi nevalide, sistemul oferă sugestii pe baza comenzilor similare.
- **Scop:** Demonstrează sistemul de sugestii pentru o experiență mai prietenoasă.

Executarea mai multor comenzi în lanț folosind pipe

- **Comanda:** hello John | sum 5 10
- **Descriere:** Execută comenzile separate prin pipe în succesiune.
- **Scop:** (dacă este implementat) Demonstrează separarea și rularea multiplă a comenzilor.

CAPITOLUL IV – MANUAL DE UTILIZARE

1. Inițializarea obiectului Parseline

Începe prin a crea o instanță a clasei Parseline în aplicația ta:

- Acest obiect va gestiona toate comenzile și interacțiunile cu utilizatorul.

---

2. Definirea comenzilor

Folosește metoda addCommand pentru a adăuga o comandă nouă. Trebuie să specifici:

- **Pattern-ul comenzii:** Numele comenzii sau expresia regulată ce definește comanda.
- **Textul de ajutor:** O descriere succintă a ceea ce face comanda.
- **Handler-ul:** O funcție care va fi executată când comanda este introdusă. Aceasta primește argumentele parsate.

Exemplu conceptual:
Adaugă o comandă hello care salută utilizatorul.

---

3. Parsarea argumentelor

Librăria parsează automat linia introdusă, separând:

- **Argumentele poziționale** (de ex. nume, numere)

Handler-ul comenzii primește un obiect ce oferă acces ușor la aceste date.

---

4. Pornirea modului interactiv (REPL)

Apelând metoda startREPL(), se pornește o buclă care afișează promptul și așteaptă comenzi de la utilizator.

- Comenzile sunt executate imediat.
- Se păstrează istoricul comenzilor introduse.
- Se oferă sugestii când comanda nu este recunoscută.
- Se poate ieși din REPL prin comanda exit.

---

5. Gestionarea istoricului

Librăria reține toate comenzile introduse în sesiune.

- Poți afișa istoricul folosind o comandă definită (ex. history).
- Acest lucru permite utilizatorului să revizuiască comenzile anterioare.

---

6. Sistemul de ajutor

- Comenzile pot avea mesaje de ajutor asociate.
- Comanda help afișează lista tuturor comenzilor sau ajutor specific pentru o comandă.

Funcționalități adiționale

- **Sugestii de comenzi:** Dacă introduci o comandă necunoscută, librăria oferă sugestii apropiate pentru a preveni erorile de tastare.

- **Separarea comenzilor prin pipe (|):** Permite rularea mai multor comenzi în aceeași linie, separate prin caracterul pipe.

Sfaturi utile

- Definirea clară a comenzilor și ajutorului ajută utilizatorii să folosească mai ușor aplicația ta.

- Verifică întotdeauna numărul și tipul argumentelor primite în handler pentru a evita erorile.

- Folosește istoricul și sugestiile pentru a îmbunătăți experiența utilizatorului.

CAPITOLUL V – CONCLUZII

Interfața produsului

Aplicația Parseline funcționează în linie de comandă și oferă o interfață interactivă tip REPL (Read-Eval-Print Loop). Utilizatorul interacționează prin introducerea de comenzi text, care sunt recunoscute și procesate în timp real. Interfața este simplă, cu un prompt clar care indică disponibilitatea de a primi comenzi, iar răspunsurile sunt afișate imediat în consolă într-un format lizibil și concis.

Datele de intrare și modul de introducere

Utilizatorul introduce comenzile ca șiruri de caractere, urmate de eventuale argumente și opțiuni. Comenzile pot conține argumente poziționale sau opțiuni cu prefix - sau --, iar librăria se ocupă de parsarea acestora în mod automat.

Comenzile pot fi simple (de ex. hello) sau pot avea argumente (ex. sum 3 5). De asemenea, pot fi introduse mai multe comenzi pe aceeași linie, separate prin caracterul pipe |.

Rezultatele oferite și formatul acestora

După executarea comenzii, rezultatul este afișat imediat în consola utilizatorului, sub forma unui text clar și structurat. Mesajele pot include:

- Răspunsuri la comenzi (ex. salutări, sume calculate)
- Liste de comenzi disponibile și ajutor detaliat

- Istoricul comenzilor introduse
- Mesaje de eroare sau sugestii pentru comenzi necunoscute

Formatul este adaptat pentru lizibilitate, cu delimitatori clari și mesaje prietenoase.

Operațiile disponibile și logica acestora

Librăria Parseline oferă următoarele funcționalități principale:

- **Definirea comenzilor personalizate:** Utilizatorul programator poate adăuga comenzi noi prin specificarea unui pattern, mesaj de ajutor și handler care primește argumentele parsate.
- **Parsarea automată a argumentelor:** Comenzile pot conține argumente și opțiuni, toate extrase și organizate într-un format ușor de utilizat.
- **Execuția comenzilor:** La introducerea unei comenzi, aceasta este căutată printre comenzile înregistrate; dacă este găsită, handler-ul asociat este executat.
- **Gestionarea istoricului:** Toate comenzile introduse sunt salvate și pot fi afișate la cerere.
- **Sistem de ajutor:** Utilizatorul poate solicita lista completă de comenzi sau ajutor pentru o comandă specifică.
- **Sugestii pentru comenzi:** Dacă se introduce o comandă necunoscută, librăria oferă sugestii bazate pe comenzile existente.
- **Separarea comenzilor multiple:** Comenzile pot fi introduse în lanțuri folosind separatorul |, fiind procesate pe rând.
- **Curățarea ecranului:** O comandă dedicată permite golirea consolei într-un mod multiplatformă.

Toate aceste operații sunt implementate prin metode dedicate din clase specializate (Parseline, ArgParser, CommandHistory, CommandHelp, CommandSuggester), respectând principiile programării orientate pe obiect și separarea clară a responsabilităților.